Inferring Change Effort from Configuration Management Databases

Todd L. Graves* and Audris Mockus+
*National Institute of Statistical Sciences and *+Bell Laboratories

ABSTRACT

In this paper we describe a methodology and algorithm for historical analysis of the effort necessary for developers to make changes to software. The algorithm identifies factors which have historically increased the difficulty of changes. This methodology has implications for research into cost drivers. As an example of a research finding, we find that a system under study was "decaying" in that changes grew more difficult to implement at a rate of 20% per year. We also quantify the difference in costs between changes that fix faults and additions of new functionality: fixes require 80% more effort after accounting for size. Since our methodology adds no overhead to the development process, we also envision it being used as a project management tool: for example, developers can identify code modules which have grown more difficult to change than previously, and can match changes to developers with appropriate expertise. The methodology uses data from a change management system, supported by monthly time sheet data if available. The method's performance does not degrade much when the quality of the time sheet data is limited. We validate our results using a survey of the developers under study: the change efforts resulting from the algorithm match the developers' opinions. Our methodology includes a technique based on the jackknife to determine factors that contribute significantly to change effort.

1 Introduction

A particularly important quantity related to software is the cost of making a change to it. Change management data contain a number of measurements on each change, such as its size and type, which one expects to be related to the effort required to make the change. In this paper we discuss our methodology and algorithms to assess the influence of these factors. This methodology has enormous potential for serving as a foundation for a variety of historical studies of effort. One example from this research is our finding that the code under study "decayed" in that changes to the code grew harder to make at a rate of 20% per year. We were also able to measure the extent to which changes which fix faults in software are more difficult than comparably sized additions of new functionality: fault fixes are approximately 80% more difficult. An important part of our methodology is the capability of assessing the amount of statistical variability of these estimated coefficients; this technique uses the jackknife [1].

Our methodology, since it adds no additional overhead to the development process, could be embodied in a project management tool. The tool would be useful for identifying modules of the code which are too costly to change, so that they might be candidates for rewriting, or for assessing developers' expertise with different modules. The tool would have wide applicability: it could be used on any version control system which stores the generic variables we discuss, such as date, size, owner, and description of the purpose of a change.

We validate our results using a number of techniques. We conducted a survey of developers in which we asked them to report how much effort they invested in making a change, and we find that the estimated efforts obtained from our algorithm are closely related to developers' opinions.

By studying effort at the level of individual changes, we are able to judge the influence of factors whose contributions are not estimable at a large project level due to aggregation. These factors might include the purpose and size of individual changes.

The next section describes the project under study as well as the quality and structure of the databases. In §4 we describe in detail the effort estimation algorithm and present the results. §5 demonstrates the high quality of the results in several ways: we validate the results using a developer survey, and compute estimates of variability of coefficients using the jackknife.

2 Change and Effort Data

We are studying changes to source code of 5ESS [2] which is a large, distributed, high availability software product. The product was developed over two decades and has tens of millions of lines of source code as well as several hundred thousand changes. The source code is mostly written in the C programming language augmented by Specification and Description (SDL) language. The tools used to process and analyze software change data are described in [3].

We use configuration management (CM) data to obtain reliable estimates of the change effort. We also use time sheet data from the financial support system (FSS) to improve those estimates. The CM data record all changes to the code, their size and content, submission time, and developer. The FSS data record only monthly effort for each developer. The link between the two databases is developer name. The change effort cannot be uniquely determined from FSS data because developers tend to work on multiple changes during most months and 14 percent of the changes start and end in different months.

The CM database has been started in 1984, while the current structure of FSS started in 1991.

2.1 Change data

The extended change management system (ECMS) and source code control system (SCCS) were used to manage the source code. Among other things, the ECMS keeps relations that map developer login to developer full name. The ECMS groups atomic changes (deltas) recorded by SCCS into logical changes referred to as Maintenance Requests (MRs). The open time of the MR is recorded in ECMS. We use time of the last delta as MR close time. We used the MR change abstract to infer the purpose of a change [4]. In addition to the three primary reasons for changes (repairing faults, adding new functionality, and improving the structure of the code; see, for example, [5]), we defined a class for changes that implement code inspection suggestions since this class was easy to separate from others and had distinct size and interval properties [4].

The SCCS database records each delta as a tuple including the actual source code that was changed, login of the developer, MR number, date, time, numbers of lines added, deleted, and unchanged.

Each logical change (MR) identifies developer login and full name. The full name can be written in numerous ways and the five most popular ones are of the form "F. M. Last", "FMLast", "F. Last", "FLast" and "login". The capitalization, periods, and spaces can vary substantially. The last form does not provide any name, but the name could be obtained by looking at other MRs that share the login.

To use FSS data we selected full developer names from FSS data, identified their logins using ECMS data, and then selected all deltas and MRs that had the appropriate login. Since identification of login using full name was not difficult (most of the names were spelled correctly in the first form shown above) and logins were unique in the considered sample (except for one developer who shortened his login in 1992) we are confident that we obtained all changes made by developers under study.

2.2 Reported Effort

In the considered project the effort was classified based on organizational accounting and reporting structure. The major development effort categories are mapped to a list of effort charging numbers, which are used by staff to log their working hours. Effort, measured in Average Technical Head Count Months (ATHCM), is recorded for each person every month broken down by charging numbers. The data are obtained from bi-weekly time sheets.

The FSS data were available from 1991, but we selected a four year period from 1994 to 1998 that had the most uniformly reported data. The FSS data before 1994 did not contain some of the developers, despite the fact that those developers were actively changing the code. We further cleaned the data by removing duplicate month/project/developer entries. Anecdotal evidence suggested that developers record their total efforts accurately but often have trouble dividing their efforts across charging numbers. Also, it was difficult reliably to link charging numbers and software changes, so we only used the total effort reported per person per month (by adding a person's effort over all charging numbers in one month).

We selected effort records of eleven of the best developers in one part of the organization. Those developers had at least seven years of experience in 1994. We later interviewed seven of those developers to learn how they perceived the effort they spent on a subset of the changes they did in the past two years (1996 and 1997). The time series of their reported effort is shown in Figure 1. The eleven developers completed 2794 changes (MRs) in the 45 month period under study.

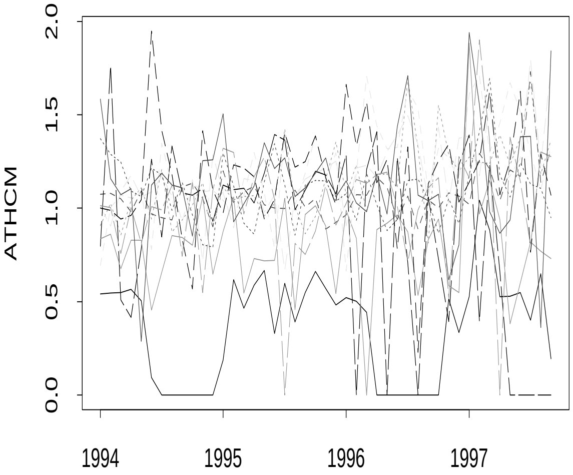

Figure 1: Reported monthly effort of 11 developers

We use the developer name (see §2.1) to relate FSS data to software changes.

3 Model framework

The following framework can help define an appropriate model and estimation procedure. Developer monthly effort is spread relatively uniformly over the month. According to [6] on average only 47 percent of developer effort in the considered organization goes to coding software changes. Hence it is natural to assume that there is significant background effort that consumes part of the monthly effort. Figure 2 shows an example of the distribution of effort over four months for one developer. The thickest line shows total developer effort. It is relatively flat indicating that total effort is spent relatively uniformly over time (we are excluding vacation time from consideration). The thin solid line shows effort not associated with changes. As changes happen, part of the total effort is diverted to those changes denoted by dotted and dashed lines. The fraction of the diverted effort depends on the type or importance of a change; for example, fault fixes tend to require a larger fraction of the effort because they have to be finished in a relatively short period of time, while new code changes take longer and are more likely to be interrupted by fault fixes or other activities not associated with the code changes.

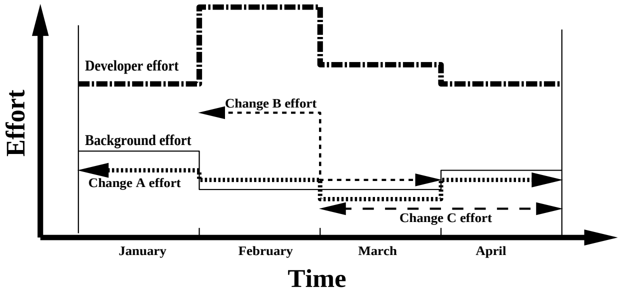

Figure 2: Time lines of effort distribution over changes

To associate the monthly effort with change effort we need to address three basic issues:

1. how the actual effort spent by a developer on a particular month corresponds to the effort reported for that month;
2. how the actual effort spent by developer on a particular change corresponds to change open, close, and atomic change times;
3. how to separate effort not related to changes.

Discussions with several developers indicated that the spent effort tends not to be shifted across months and that the amount of work on an MR performed before its recorded open time or after its last delta time is insignificant. We did not address the issue of whether the effort is uniformly distributed over the time the MR is open. The proportion of background effort was obtained from [6] who studied the same project four years earlier. Since our method is scale invariant the proportion can be applied to the final result if the absolute value of change effort is desired.

To use monthly reported effort data we took the simplest approximation, in which all changes open during a month share part of the effort spent during the month. That proportion is determined using the modeling technique described in the next section. As the starting point for that procedure we chose to distribute effort equally among MRs done by the same developer in one month. The results are robust to large changes in starting values, so we did not attempt to specify starting values more precisely. Example starting values for one developer are shown in Table 1. The table represents the effort shown in Figure 2. The total effort for Change A in the example was 2.1 ATHCM, 0.9 ATHCM for Change B and 0.7 ATHCM for Change C.

Table 1: Starting Values for Iterative Fitting Procedure.

	Jan	Feb	Mar	Apr	Total
is Change A open	yes	yes	yes	yes	
is Change B open	no	yes	yes	no	
is Change C open	no	no	yes	yes	
Change A effort	0.8	0.6	0.3	0.4	2.1
Change B effort	0	0.6	0.3	0	0.9
Change C effort	0	0	0.3	0.4	0.7
reported effort	0.8	1.2	0.9	0.8	3.7

4 Effort Estimation Methodology and Results

In this section we discuss the procedure we follow to identify factors that affect effort, given the sort of data discussed earlier. We first describe the procedure itself, which is the main contribution. The regression model quantifies factors that predict effort and illustrates one application of this methodology. In §5.3 we provide advice about how to proceed when time sheet data are unavailable.

4.1 Iterative algorithm to estimate change effort

Consider the data corresponding to a single developer; the generalization to multiple developers is natural but initially confusing. These data may be thought of as a two-way table, as in the numeric entries in Table 1, in which rows correspond to MRs and columns to months. The reported monthly efforts for the developer are the column sums of this table. The cells of the table represent the amount of effort that the developer expends on a particular MR in that month. They are not observable, except that a majority of cells contain zeroes because the MR was not open during that month. The objective is to obtain the row sums, the total effort associated with an MR. To accomplish that we use a number of quantities from the change management system which are closely related to MR efforts. We will use an iterative algorithm to impute values of the cells and the row sums and use the reported effort to calibrate and improve these imputed values. To initialize the procedure we spread the monthly effort evenly across the MRs that were open during that month. We tried a number of other ways to to initialize, and obtained very similar results in all cases.

After the initial guess the algorithm contains four steps that are repeated until convergence of the error measure (10 iterations were always enough to obtain convergence).

1. Compute row sums to obtain estimates of total MR efforts.
2. Fit a regression model, such as in Equation 1, of imputed MR effort on the factors that predict MR effort.
3. Rescale the rows in the imputed monthly MR effort table so that the new row sums are equal to the regression's fitted values.
4. Rescale the columns of the table so that the column sums are equal to the observed monthly efforts.

We prefer to use a generalized linear model in the second step since we can then guarantee that fitted values are positive.

4.2 Valuable predictors of change effort

We found four types of variables to be critical contributors to estimated change effort. Not surprisingly, the size of a change is important. Size can be measured by the number of files changed by an MR, the sum of the numbers of lines added and deleted in the MR, as well as many other size measures, but the measure we found to be slightly better than these is the number of deltas in the MR.

Another useful predictor is the developer making the change. Our model found that one developer tends to expend 2.75 times as much effort as another developer to make a comparable change, and although we did not find differences between developer effects to be statistically significant, we thought it was important to leave this variable in the model. In other studies, developer productivity has varied dramatically, see, for example, [7]. Developer effects would be significant in studies which do not restrict attention to developers that have similar experience. However, even if we had found significant differences, their interpretation would be problematic. Not only could they appear because of differing developer abilities, but the seemingly less productive developer could be the expert on a particularly difficult area of the code, or that developer could have more extensive duties outside of writing code.

We found that the purpose of the change (as estimated using the techniques of [4]) also had a strong effect on the effort required to make a change. Bug fixes are more difficult than comparably sized additions of new code by approximately a factor of 1.8. Bug fixes are more difficult than additions of new code even before allowance for the size of the change, although additions of new code are generally significantly larger.

The last significant predictor of the change effort was the date the change was opened. We were interested to see if there was evidence that the code was getting harder to change, or *decaying*, as discussed in [8, 9, 10]. There was statistically significant evidence of a decay effect: we estimated that in our data, a change begun a year later than an otherwise similar change would require 20% more effort than the earlier change.

The specific model we fit in the modeling stage of the algorithm was a generalized linear model ([11]) of effort. MR effort was modeled as having a Poisson distribution with mean given below.[1]

$$E(\text{effort}) = 10^{-1} \times \alpha_{DEV} \times \beta_{TYPE} \times \text{Size}^{\gamma} \times \theta^{\text{DATE}}. \quad (1)$$

Here the estimated developer coefficients, the α's, ranged from 0.35 to 0.96, with no statistically significant evidence that the developers were different from each other (according to standard deviation obtained via jackknife as in 5.2). We defined β_{NEW}, the coefficient for changes that add new functionality, to be unity to make the problem identifiable, then estimated the remaining type coefficients to be 1.8 for fault fixes, 1.4 for restructuring and cleanup work, and 0.8 for inspection rework.

The value of γ, which is the power of the number of files changed which affects effort, we estimated to be 0.26, and its jackknife standard error estimate is 0.06. Since $\gamma < 1$, for these data, the difficulty of a change increases sublinearly with the change's size. Large changes seem to be mostly straightforward, while small changes may require large initial investments of time to find where to make the change.

θ was estimated to be 1.21, and its natural logarithm had a standard error of 0.07, so that a confidence interval for θ calculated by taking two standard errors in each direction of the estimated parameter is $1.21 \exp\{\pm 2 \times 0.07\} = (1.05, 1.39)$. This implies that a similar change becomes 5% to 39% harder to make in one year.

Surprisingly the interval required to complete the change was not a significant predictor of change effort. Probably the size of the change is a better measure of the effort than the interval. Also, organizational rules required that fault fixes, which are often difficult, be completed rapidly.

5 Validation

In this section we give details of how we confirm our results. We have used four methods: first, we conducted interviews of developers and asked them how difficult some of the changes they had made were. Second, we used the *jackknife*, a resampling type method, to obtain estimates of the uncertainty of the estimated coefficients. Then, we tested the sensitivity of the results to the variation of the reported effort. Finally, we note that the estimated coefficients have values which are roughly what one would expect.

[1] We strictly speaking do not assume a Poisson distribution because effort values need not be integers. The only critical part of the Poisson assumption is that the variance of a random effort is proportional to its mean.

5.1 Developer survey

Seven of the developers were surveyed to assess their opinions on the change effort. The survey asked them to “rate the difficulty of the change to do in terms of effort and time relative to your experience”. Ten to thirty randomly selected changes completed by the developer over past two years were on the survey. Developers were asked to rate changes into one of three levels of difficulty: easy, medium, and hard. More details regarding the survey are in [4].

Since it is possible that some developers might be more reluctant to describe changes as difficult, it is important to adjust for the differences in developer opinions. Also the difficulty levels might not be linearly related to effort, i.e., the difference between easy and medium might not be the same as the difference between medium and hard.

To answer the first question we fitted a generalized linear model

$$E(\text{effort}_{\text{assessed}}) = \text{developer} + \text{effort}_{\text{fitted}}$$

The developer effect should remove the differences between the developer opinions. Both effects were highly significant (p-value much less than 0.01). This shows that the estimated effort predicts developer assessment and it also indicates that there were substantial differences between what each developer perceived as hard.

To check if the developer assessment is linear we provide boxplots of the imputed values broken by developer assessment in Figure 3. We see that imputed effort clearly discriminates the “hard” changes but does not differentiate between “easy” and “medium” changes.

5.2 Jackknife

The jackknife (see, for instance, [1]) supplies a method of estimating uncertainties in estimated parameters by rerunning the estimation algorithm once for each data point in the sample, each time leaving one of the data points out. A collection of estimated parameters results, and the degree of difference between these parameters is related to the sampling variability in the overall estimate. For example, suppose that the statistic of interest is denoted by $\hat{\theta}$, which was computed from a dataset of size n. What is required is an estimate of the standard error of $\hat{\theta}$. For $i = 1, 2, \ldots, n$, compute $\hat{\theta}_{(i)}$ using the data set with the ith observation deleted. Set $\hat{\theta}_{(\cdot)} = n^{-1} \sum_{i=1}^{n} \hat{\theta}_{(i)}$. The jackknife estimate of standard error of $\hat{\theta}$ is

$$\left\{ \frac{n-1}{n} \sum_{i=1}^{n} (\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)})^2 \right\}^{1/2} .$$

The jackknife and the bootstrap (for which see [13]) are examples of resampling techniques, which aim to extract additional information from data sets by the process of constructing new data sets by sampling from the observed data. A bootstrap replication selects n data points from a sample of size n by sampling with replacement; the bootstrap repeats this process several hundred times and is generally preferred to the jackknife, computational time permitting. Owing to the time consuming iterative algorithm, we preferred the jackknife since it required only eleven iterations. For the purposes of resampling, there are only eleven datapoints in the problem at hand. We have one point for each developer, because omitting some of a developer’s changes leaves some months in which the total effort of the MRs is less than the observed monthly effort. Omitting some months will break up MRs that span those months and the total effort for such an MR cannot be calculated.

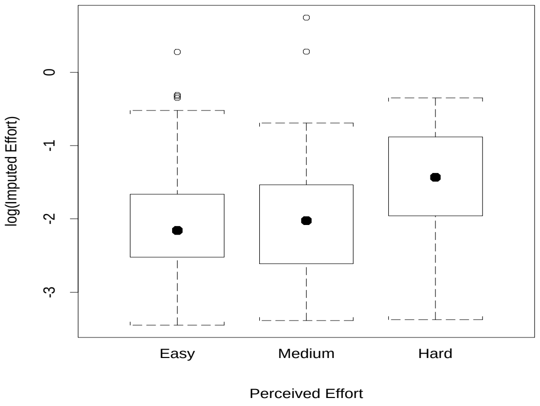

Figure 3: Boxplots of imputed effort broken by developer assessment. In a boxplot a data set is represented by a box whose height spans the central 50% of the data. The upper and lower ends of the box marks the upper and lower quartiles. The data’s median is denoted by a dot within the box. The dashed vertical lines attached to the box indicate the tails of the distribution; they extend to the standard range of the data (1.5 times the inter-quartile range). All other detached points are “outliers.”[12].

Jackknife estimates of standard error are 0.06 for γ, 0.20 for the natural logarithm of β_{bug}, values between 0.19 and 0.30 for the natural logarithms of the developer coefficients, and 0.07 for the natural logarithm of θ.

5.3 Sensitivity to reported effort

To test the sensitivity of the results to the reported effort data and to check the validity of the methodology when fine monthly efforts are not available we used yearly developer efforts, which we believe are very reliable. We distributed yearly developer efforts uniformly over 12 months. Then we fitted the model to determine factors that contribute to the change effort. The resulting estimated coefficients are close to the ones obtained using finer monthly data. For example, the estimate of γ changed from 0.26 to 0.25, and the value of θ was the same to three decimal places. The only substantial change was the coefficient for a developer who stopped making changes for two extended periods of six and seven months. As a consequence of this, this developer's coefficient changed by about 22%.

We believe that effort is distributed relatively uniformly over time (given that a person has quite restrictive limits on the amount of effort she can spend in a limited amount of time) and hence it is not essential to have very detailed effort measurements over time to obtain principal factors that contribute to change effort.

6 Discussion

The proposed methodology for assessing the principal predictors of code change effort from the historic software data gives ways to impute the effort associated with every change and a model to determine significant factors affecting change effort. We determined that the purpose of a change is at least as important as its size: since bug fixes are 1.8 times as difficult as additions of new code and since the size of a change enters the model as the 0.26 power of the number of deltas in the change, an addition of new code would have to include about ten times as many deltas as a fault fix for it to require as much effort. This could not be established using more aggregate project effort because most projects mix changes of different types. Also, we discovered a significant increase in effort to make a similar change later in the period. We estimate the rate of increase of required effort to be about 20% per year, so this is the most prominent indicator of code decay [10] that we are aware of. Our estimates of developer coefficients found that one developer required 2.75 times as much effort as another developer to perform comparable changes, but our jackknife estimates of standard error suggested that the differences between developers were not statistically significant. We would likely have found larger differences between developers had we not restricted attention to experienced developers.

We focus on methodology implementable with minimal or no overhead to developers. This leads us to use source code version control data, of a sort present in many software development organizations, and to avoid collecting additional information from the developers. Since the version control data are not designed to answer questions about the software development process, we developed and used automatic algorithms to extract the information hidden in the data. We first classified the changes according to their purposes and then imputed the effort associated with each change. The imputation method also produces the key factors contributing to the change effort.

Since the methodology is based on variables from version control data which are widely collected, it should be easily replicated. We spent a significant amount of effort validating the results using a developer survey and statistical techniques. We found that the results are not very sensitive to how reported developer effort is distributed over time. This may be due to the fact that a person has natural limits on the effort he can spend over a limited time. The important ramification is that the change effort can be estimated based solely on version control data and without the reported developer effort. We believe that the methodology can be successfully applied in any project that has one or more years of version control data.

In the future we intend to use the estimated effort in tools to assess code decay and to find developers expert in certain areas of the system. In the immediate future we intend to validate the methodology by applying it in a different organization.

ACKNOWLEDGEMENTS

We thank George Schmidt, Harvey Siy, Mark Ardis, David Weiss, Alan Karr, Iris Dowden, and interview subjects for their help and valuable suggestions. This research was supported in part by NSF grants SBR-9529926 and DMS-9208758 to the National Institute of Statistical Sciences.

REFERENCES

[1] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1982.

[2] K. Martersteck and A. Spencer, “Introduction to the 5ESS(TM) switching system,” *AT&T Technical Journal*, vol. 64, pp. 1305–1314, July-August 1985.

[3] A. Mockus, S. G. Eick, T. Graves, and A. F. Karr, “Infrastructure for the analysis of software data,” Submitted to Software Engineering and Knowledge Engineering.

[4] A. Mockus and L. G. Votta, “Identifying reasons for software changes using historic databases,” Submitted to ACM Transactions on Software Engineering and Methodology.

[5] E. B. Swanson, “The dimensions of maintenance,” in *2nd Conf. on Software Engineering*, (San Francisco, California), pp. 492–497, 1976.

[6] D. E. Perry, N. A. Staudenmayer, and L. G. Votta, “People, organizations, and process improvement,” *IEEE Software*, pp. 36–45, July 1994.

[7] B. Curtis, “By the way, did anyone study any real programmers?,” in *Empirical Studies of Programmers: Papers presented at the First Workshop on Empirical Studies of Programmers*, (Washington, DC), pp. 256–262, 1986.

[8] L. A. Belady and M. M. Lehman, “A model of large program development,” *IBM Systems Journal*, pp. 225–252, 1976.

[9] D. L. Parnas, “Software aging,” in *Proceedings 16th International Conference On Software Engineering*, (Los Alamitos, California), pp. 279–287, IEEE Computer Society Press, 16 May 1994.

[10] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, “Does code decay? assessing the evidence from change management data,” *Submitted to IEEE Trans. Soft. Engrg.*, 1998.

[11] P. McCullagh and J. A. Nelder, *Generalized Linear Models, 2nd ed.* New York: Chapman and Hall, 1989.

[12] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey, *Graphical Methods For Data Analysis*. Chapman & Hall, 1983.

[13] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. New York: Chapman and Hall, 1993.